A Theory of First-Order Built-in's of Prolog

Krzysztof R. Apt, *† Elena Marchiori, *‡ Catuscia Palamidessi §

Abstract

We provide here a framework for studying Prolog programs with various built-in's that include arithmetic operations, and such metalogical relations like *var* and *ground*. To this end we propose a new, declarative semantics and prove completeness of the Prolog computation mechanism w.r.t. this semantics. Finally, we provide a method for proving termination of Prolog programs with built-in's which uses this semantics. The method is shown to be modular.

Note: This research was done during the second and third authors' stay at Centre for Mathematics and Computer Science, Amsterdam. The work of K.R. Apt was partly supported by ESPRIT Basic Research Action 3020 (Integration). The work of C. Palamidessi was partly supported by ESPRIT Basic Research Action 3020 and by the Italian CNR (Consiglio Nazionale delle Ricerche). The work of E. Marchiori was partly supported by the Italian CNR under Grant No. 89.00026.69.

1 Introduction

1.1 Motivation

Theory of logic programming allows us to treat formally only pure Prolog programs, that is those whose syntax is based on Horn clauses. Any formal treatment of more realistic Prolog programs has to take into account the use of various built-in's. Some of them, like arithmetic relations, seem to be trivial to handle, as they simply refer to some theory of arithmetic. However, the restrictions on the form of their arguments (like the requirement that both arguments of $<$ should be ground) cause complications which the theory of logic programming does not properly account for. In particular, in presence of arithmetic relations the independence of the refutability from the selection rule fails, as the goal $\leftarrow x = 2, 1 < x$ shows.

Further, the use of metalogical relations (like *var*, *ground*) leads to various additional problems. Clearly, *var* cannot be handled using the traditional semantics based on first-order logic because $var(x)$ is true whereas some instances of it are not. In presence of *nonvar* another complication arises: the well-known Lifting Lemma (see Lloyd [Llo87]) needed to prove completeness of the SLD-resolution does not hold — for a non-variable term t, the goal $\leftarrow nonvar(t)$ can be refuted whereas its more general version $\leftarrow nonvar(x)$ cannot.

Finally, study of termination of Prolog programs in presence of the above built-in's calls for some new insights. For example, the program `list`

*Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands,

†Faculty of Mathematics and Computer Science, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands,

‡Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italy,

§Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy.

```
list([]) ←.
list([X|Xs]) ←
  nonvar(Xs), list(Xs).
```

which recognizes a list, always terminates, whereas its pure Prolog counterpart obtained by dropping the atom $nonvar(Xs)$ may diverge. As a result the methods developed to reason about termination of pure Prolog programs (see Apt and Pedreschi [AP91] for a short overview) cannot be used here.

The aim of this paper is to provide a systematic account of the class of the above mentioned built-in's of Prolog. This class includes the arithmetic relations (like $:=, <$ etc.) and some metalogical relations (like $var, ground$ etc.). To distinguish them from those built-in's which refer to clauses and goals (like *call* and *assert*), we call them *first-order* built-in's. Hence the title.

In Section 2 we introduce a new declarative semantics and prove a completeness result connecting it with the Prolog computational mechanism. In Section 3 we show how this semantics can be used to prove termination of Prolog programs with first-order built-in's. We also show how termination proofs can be constructed in a modular way.

We are aware of two other approaches to define the meaning of Prolog programs with built-in's, namely that of Börger [Bör89] based on so-called dynamic algebras, and that of Deransart and Ferrand [DF87] based on an abstract interpreter. Their aim is to provide semantics for the complete Prolog language whereas ours is to extend the declarative semantics to Prolog programs with built-in's so that one can reason about such programs. In this respect our approach has the same aim as that of Hill and Lloyd [HL88] where all metalogical features of Prolog are represented in a uniform way by means of a representation of the object level in the meta-level, reminiscent of the Gödelization process in Peano arithmetic. In contrast, we are not aware of any work on termination of Prolog programs with built-in's.

1.2 Preliminaries

In what follows we study logic programs extended by various built-in relations. We call the resulting objects *Prolog programs*, or simply *programs*. Prolog programs are executed by means of the *LD-resolution*, which consists of the usual SLD-resolution combined with the leftmost selection rule, that is appropriately extended to deal with the built-in relations.

We often manipulate various sets of variables. In general $\mathbf{x}, \mathbf{y}$ stands for sequences of different variables. Sometimes we identify such sequences with sets of variables. Given a substitution η and a set of variables $\mathbf{x}$ we denote by $\eta \mid \mathbf{x}$ the substitution obtained from η by restricting its domain, $Dom(\eta)$, to $\mathbf{x}$. By $Ran(\eta)$ we denote the set of variables that appear in the terms of the range of η. A *renaming* is a substitution that is a permutation of the variables constituting its domain.

Given an expression (term, atom, goal,...) or a substitution E we denote the set of variables occurring in it by $Var(E)$. We often write $\eta \mid E$ to denote $\eta \mid Var(E)$. The set of all variables is denoted by Var. Atoms of the form $p(\mathbf{x})$ where p is a relation are called *elementary atoms* and atoms containing a built-in relation are referred to as *built-in atoms*. Finally, atoms containing a relation used in a head of a clause of a program P are said to be *defined in* P.

In the context of logic programs, or more generally, Prolog programs, it is convenient to treat sequences of atoms as conjunctions (sometimes called conjuncts). By the *length* of such a conjuction we mean the number of its atoms. Usually, $\mathbf{A}, \mathbf{B}$ denote such conjunctions.

It is convenient to associate with each pair of atoms or pair of terms that unify a unique idempotent and relevant mgu in the sense of Apt [Apt90, page 502]. Given such a pair A, B

we denote it by $mgu(A, B)$. Recall that an mgu η of A and B is *idempotent* if $\eta\eta = \eta$ and is *relevant* if $Var(\eta) \subseteq Var(A, B)$. The relation *more general than* defined on pairs of atoms, terms or substitutions is denoted by $\leq$.

The rest of the used notation is more or less standard and essentially follows Lloyd [Llo87]. In particular c.a.s. stands for *computed answer substitution.*

2 The declarative semantics

2.1 Motivation

In this section we define a declarative semantics appropriate to describe the operational behaviour of Prolog programs. First, let us see why it is impossible to achieve this goal by simply modifying one of the usually considered declarative semantics.

The standard declarative semantics, based on the (ground) Herbrand models due to van Emden and Kowalski [vEK76], is clearly inadequate to deal with first-order built-in's. Indeed, in this semantics, in a given interpretation, if an atom is true then all its ground instances are. However, for every ground term t, $var(t)$ should be false in every model whereas $var(x)$ should be true. Therefore we say that *var* is a *non-monotonic relation.*

We conclude that any declarative modeling of non-monotonic relations requires an explicit introduction of non-ground atoms in the Herbrand interpretations, in order to define the truth value of an atom independently from its ground instances. The non-ground Herbrand semantics proposed by Clark [Cla79] (called C-semantics in [MP89]) is however not adequate, because it is monotonic. Namely, if $A(x)$ is true in an interpretation, then also $A(t)$ is true, for every t, in the same interepretation.

In presence of built-in relations like *nonvar*, another problem arises: the goal $\leftarrow nonvar(x)$ fails whereas for every non-variable term t, the goal $\leftarrow nonvar(t)$ succeeds. Therefore we say that *nonvar* is a *non-down-monotonic relation.* Due to the presence of non-down-monotonic relations the Lifting Lemma (see Lloyd [Llo87]) does not hold for Prolog programs. In particular, for the program

```
p(X) ← nonvar(X).
```

for every non-variable term t, the goal $\leftarrow p(t)$ has a refutation, whereas $\leftarrow p(x)$ fails.

This example rules out the S-semantics of Falaschi et al. [MP89] in which the meaning of a relation p is identified with the set of computed answer substitutions η of the goal $\leftarrow p(\mathbf{x})$ - in a sense, the post-conditions which are verified *after* the possible succesful computations of the goal $\leftarrow p(\mathbf{x})$. We also need a pre-condition, i.e. information about the substitution θ by which the atom $p(\mathbf{x})$ is instantiated *before* starting the computation. A possible way to do it is by enriching the domain with another component, thus explicitly representing the substitution used before starting the computation.

2.2 Θ-semantics

The considerations made in the previous section lead us to consider objects of the form $\langle \theta, p(\mathbf{x}), \eta \rangle$, where θ represents the *pre-substitution* (or *input substitution*) and η represents the *post-substitution* (or *output substitution*) for the goal $\leftarrow p(\mathbf{x})$. For technical convenience we equivalently represent these triples as pairs of the form $\langle A, \eta \rangle$, where A is the atom obtained by applying the input substitution θ to the elementary atom $p(\mathbf{x})$, i.e. $A = p(\mathbf{x})\theta$.

Of course, we can restrict our attention to pairs $\langle A, \eta \rangle$ in which η does not affect the variables that do not appear in A.

First, we deal with built-in relations. For any such relation p we stipulate a set $[\![p]\!]$ of pairs defining its operational behaviour. We list here some cases. In the definition below, "=" is the well-known built-in standing for "is unifiable with".

$[\![var]\!] = \{\langle var(x), \epsilon \rangle \mid x \in Var\}$,

$[\![nonvar]\!] = \{\langle nonvar(s), \epsilon \rangle \mid s \notin Var\}$,

$[\![=]\!] = \{\langle s = t, \eta \rangle \mid \eta = mgu(s,t)\}$,

$[\![>]\!] = \{\langle s > t, \epsilon \rangle \mid s, t \text{ are integers and } s > t\}$,

$[\![constant]\!] = \{\langle constant(a), \epsilon \rangle \mid a \text{ is a constant}\}$,

$[\![compound]\!] = \{\langle compound(s), \epsilon \rangle \mid s \text{ is a compound term}\}$,

$[\![functor]\!] = \{\langle functor(t, f, n), \eta \rangle, t = (f\eta)(t_1, \ldots, t_{n\eta}) \mid$
$Dom(\eta) \subseteq \{f, n\}$, $n\eta$ is a natural number and for some $t_1, \ldots, t_{n\eta}$, or
$Dom(\eta) = \{t\}$ and $t\eta = f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are fresh variables$\}$,

$[\![:=]\!] = \{\langle x := s, \{x/t\} \rangle \mid x \in Var,\ s \text{ is a ground arithmetic expression with value } t\}$,

$[\![arg]\!] = \{\langle arg(n, s, t), \eta \rangle \mid Dom(\eta) \subseteq \{t\},\ n \text{ is a natural number and}$
$t\eta \text{ is the } n-th \text{ argument of } s, \text{ or} Dom(\eta) = \{s_n\} \text{ and } s_n\eta = t\}$.

We assume that the set of pairs associated with a built-in relation describes *correctly* its operational behaviour, in the following sense.

Definition 2.1 Let A be an atom with a built-in relation p. Then for every conjunction $\mathbf{B}$, the goal $\leftarrow \mathbf{B}\eta$ is a resolvent of $\leftarrow A, \mathbf{B}$ iff $\langle A, \eta \rangle \in [\![p]\!]$. □

Next, we consider atoms defined by the program. Given a conjunct $\mathbf{A}$ of atoms we denote by $l(\mathbf{A})$ its length. If $l(\mathbf{A}) = 0$ we denote $\mathbf{A}$ by *true*. First we introduce the following generalization of Herbrand base and Herbrand intepretation.

Definition 2.2 (Θ-domain and Θ-interpretation) Let P be a Prolog program.

- The Θ-*base* Θ_P of P is the set of all pairs $\langle A, \eta \rangle$, where A is an atom defined in P, and η is a substitution s.t. $Dom(\eta) \subseteq Var(A)$.
- A Θ-interpretation $\mathcal{I}$ of P is a subset of the Θ-base Θ_P. □

To define the truth in Θ-interpretations we have to model appropriately the proof theoretic properties of the computed answer substitutions. To this end it is important to reflect on them first.

Definition 2.3 Let $\mathbf{A}, \mathbf{B}$ be conjuncts and let θ, σ be substitutions. We say that $(\mathbf{A}, \mathbf{B}, \theta, \sigma)$ is a *good tuple* if the following conditions are satisfied:

- $Ran(\theta) \cap Var(\mathbf{B}) \subseteq Var(\mathbf{A})$
  (the variables introduced by θ that occur in $\mathbf{B}$ also occur in $\mathbf{A}$),
- $Ran(\sigma) \cap (Var(\mathbf{A},\mathbf{B}) \cup Ran(\theta)) \subseteq Var(\mathbf{B}\theta)$
  (the variables introduced by σ that occur in $\mathbf{A},\mathbf{B}$ or in $Ran(\theta)$ also occur in $\mathbf{B}\theta$). □

The importance of this, admittedly esoteric, notion is revealed by the following lemma.

Lemma 2.4 (Good Tuple) *Consider a goal* $\leftarrow \mathbf{A},\mathbf{B}$. *Then* η *is a c.a.s. of* $P \cup \{\leftarrow \mathbf{A},\mathbf{B}\}$ *iff for some* θ *and* σ

- θ *is a c.a.s. of* $P \cup \{\leftarrow \mathbf{A}\}$,
- σ *is a c.a.s. of* $P \cup \{\leftarrow \mathbf{B}\theta\}$,
- $\eta = (\theta\sigma)\,|\,(\mathbf{A},\mathbf{B})$,
- $(\mathbf{A},\mathbf{B},\theta,\sigma)$ *is a good tuple.* □

This lemma shows that the c.a.s.'s for a compound goal $\leftarrow \mathbf{A},\mathbf{B}$ cannot be obtained by simply composing each c.a.s. θ for $\leftarrow \mathbf{A}$ with each c.a.s. σ for $\leftarrow \mathbf{B}\theta$. The notion of a good tuple formalizes the conditions that θ and σ have to satisfy. Both conditions of Definition 2.3 of Good Tuple are needed.

Consider for example the program P: p(Y) $\leftarrow$. and the goal $G = \leftarrow p(X), p(Y)$. Then $\theta = \{X/Y\}$ is a c.a.s. for $\leftarrow p(X)$ and $\sigma = \epsilon$ is a c.a.s. of $P \cup \{\leftarrow p(Y)\theta\}$ but $(\theta\sigma)\,|\,G = \{X/Y\}$ is not a c.a.s. of $P \cup \{G\}$. This shows that the first condition in Definition 2.3 of good tuple is needed.

Now $\theta = \epsilon$ is also a c.a.s. for $\leftarrow p(X)$, $\sigma = \{Y/X\}$ is a c.a.s. of $P \cup \{\leftarrow p(Y)\theta\}$ (rename the clause with $\{Y/X\}$) but $(\theta\sigma)\,|\,\mathbf{G} = \{Y/X\}$ is not a c.a.s. of $P \cup \{\mathbf{G}\}$. This shows that the second condition in Definition 2.3 of Good Tuple is needed.

Since we want to model the meaning of a conjunct w.r.t. a post-substitution η in such a way that a precise match with the procedural semantics is maintained, the notion of a good tuple will be crucial also for the semantic considerations.

The next step is dictated by the simplicity considerations. We shall restrict our attention to Prolog programs in a certain form. Then, after proving soundness and completeness for these programs, we shall return to the general case.

Definition 2.5 (Homogeneous Programs)

- A Prolog clause is called *homogeneous* if its head is an elementary atom.
- A Prolog program is called *homogeneous* if all its clauses are homogeneous. □

We now define truth in Θ-interpretations for homogeneous programs. It relies on the notion of a good tuple.

Definition 2.6 (Truth in Θ-interpretations) Let $\mathcal{I}$ be a Θ-interpretation of a homogeneous Prolog program P.

The *truth* of a conjunct $\mathbf{A}$ in $\mathcal{I}$ w.r.t. a (post-)substitution η, denoted by $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$, is defined by induction on $l(\mathbf{A})$.

- $l(\mathbf{A}) = 0$. Then $\mathbf{A} = \mathit{true}$.
  $\mathcal{I} \models \langle \mathit{true}, \eta \rangle$ iff $\eta = \epsilon$.
- $l(\mathbf{A}) = 1$. Then $\mathbf{A} = A$ for an atom A.
  $\mathcal{I} \models \langle A, \eta \rangle$ iff $\langle A, \eta \rangle \in [\![p]\!]$, where A is a built-in atom with the relation symbol p,
  $\mathcal{I} \models \langle A, \eta \rangle$ iff $\langle A, \eta \rangle \in \mathcal{I}$, where A is defined in P.
- $l(\mathbf{A}) > 1$. Then $\mathbf{A} = A, \mathbf{B}$ for an atom A and a non-empty conjunct $\mathbf{B}$.
  $\mathcal{I} \models \langle A, \mathbf{B}, \eta \rangle$ iff there exist θ, σ s.t.
  - $\mathcal{I} \models \langle A, \theta \rangle$,
  - $\mathcal{I} \models \langle \mathbf{B}\theta, \sigma \rangle$,
  - $\eta = (\theta\sigma)|(A, \mathbf{B})$,
  - $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple.

The *truth* of a homogeneous clause $H \leftarrow \mathbf{B}$ of P in $\mathcal{I}$, denoted by $\mathcal{I} \models H \leftarrow \mathbf{B}$, is defined as follows.

- $\mathcal{I} \models \langle H \leftarrow \mathbf{B}, \eta \rangle$ iff for all θ s.t. $\mathit{Dom}(\theta) = \mathit{Var}(H)$ and $\mathit{Ran}(\theta) \cap \mathit{Var}(\mathbf{B}) = \emptyset$,
  $\mathcal{I} \models \langle \mathbf{B}\theta, \eta \rangle$ implies $\mathcal{I} \models \langle H\theta, \eta|(H\theta) \rangle$,
- $\mathcal{I} \models H \leftarrow \mathbf{B}$ iff for all η, $\mathcal{I} \models \langle H \leftarrow \mathbf{B}, \eta \rangle$.

$\mathcal{I}$ is a Θ-*model* of P iff all variants of the clauses of P are true in $\mathcal{I}$. □

2.3 Θ-semantics and LD-resolution

The next step is to show that LD-resolution is correct w.r.t. the Θ-semantics. The proof relies on the Good Tuple Lemma 2.4. It is convenient to assume that whenever in the LD-resolution step the selected atom A is unified with the head H of the input clause where H is a pure atom, then the mgu θ of A and H is s.t. $\mathit{Dom}(\theta) = \mathit{Var}(H)$. Thus $A = H\theta$. By the length $l(\xi)$ of a derivation ξ we mean here the number of its goals.

Theorem 2.7 (Soundness I) *Let P be a homogeneous Prolog program and $\mathbf{A}$ a conjunct. If η is a c.a.s. for $P \cup \{\leftarrow \mathbf{A}\}$ then for any Θ-model $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$.*

Proof. Fix a Θ-model $\mathcal{I}$ of P. Let ξ be a LD-refutation of $P \cup \{\leftarrow \mathbf{A}\}$ with c.a.s. η. We prove the claim by induction on the length $l(\xi)$ of ξ. Three cases arise.

Case 1 $l(\mathbf{A}) = 0$. Then $\mathbf{A} = \mathit{true}$ and $\eta = \epsilon$, so the claim follows directly by Definition 2.6.

Case 2 $l(\mathbf{A}) = 1$. Then $\mathbf{A} = A$ for an atom A.
If A is a built-in atom, then the claim follows directly by Definitions 2.1 and 2.6. If A is defined in P, then consider the resolvent $\mathbf{B}\theta$ of $\leftarrow A$ in ξ obtained using the input clause $H \leftarrow \mathbf{B}$ and mgu θ. H is an elementary atom and by the standardization apart A and $H \leftarrow \mathbf{B}$ have no variable in common, so

$$\mathit{Dom}(\theta) = \mathit{Var}(H),\ \mathit{Ran}(\theta) \cap \mathit{Var}(\mathbf{B}) = \emptyset, \tag{1}$$

and

$$A = H\theta. \tag{2}$$

Let η' be the c.a.s. for $P \cup \{\leftarrow \mathbf{B}\theta\}$ computed by the suffix ξ' of ξ starting at $\leftarrow \mathbf{B}\theta$. Then

$$\eta = (\theta\eta') \,|\, A. \tag{3}$$

We have $l(\xi') = l(\xi) - 1$, so by the induction hypothesis $\mathcal{I} \models \langle \mathbf{B}\theta, \eta' \rangle$. But $\mathcal{I}$ is a model of P, so $H \leftarrow \mathbf{B}$ is true in $\mathcal{I}$ and consequently by (1) and Definition 2.6 $\mathcal{I} \models \langle H\theta, \eta' \,|\, H\theta \rangle$. Thus by (2) $\mathcal{I} \models \langle A, \eta' \,|\, A \rangle$. However, A and H have no variable in common, so by (1) $\theta \,|\, A = \epsilon$ and consequently by (3) $\eta = (\theta\eta') \,|\, A = \eta' \,|\, A$. So we proved $\mathcal{I} \models \langle A, \eta \rangle$.

Case 3 $l(\mathbf{A}) > 1$. Then $\mathbf{A} = A, \mathbf{B}$ for an atom A and a non-empty conjunct $\mathbf{B}$.
By the Good Tuple Lemma 2.4 there exist θ and σ s.t. $\eta = (\theta\sigma) \,|\, \mathbf{A}$ and

(i) $P \cup \{\leftarrow A\}$ has an LD-refutation ξ_1 with c.a.s. θ,

(ii) $P \cup \{\leftarrow \mathbf{B}\theta\}$ has an LD-refutation ξ_2 with c.a.s. σ,

(iii) $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple.

Moreover, by the proof of this lemma it follows that we can choose ξ_1, ξ_2 to be subderivations of ξ. Then $l(\xi_1) < l(\xi)$ so by the induction hypothesis

$$\mathcal{I} \models \langle A, \theta \rangle. \tag{4}$$

Also $l(\xi_2) < l(\xi)$, so by the induction hypothesis

$$\mathcal{I} \models \langle \mathbf{B}\theta, \sigma \rangle. \tag{5}$$

Thus by (iii), (4) and (5) we get $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$ by Definition 2.6. □

In order to prove the converse of Theorem 2.7 it is helpful to consider a special Θ-model representing all Θ-models, in the sense that a conjunction is true in it (w.r.t. a given post-substitution) iff it is true in all Θ-models.

The Θ-interpretations are naturally ordered by the set inclusion. In this ordering the least Θ-interpretation is $\emptyset$ and the greatest one is Θ_P. Analogously to standard Herbrand models, the Θ-models are closed under arbitrary intersections, from which we deduce the existence of the least Θ-model.

Theorem 2.8 *Let P be a homogeneous program and $\mathcal{M}$ be a class of Θ-models of P. Then $M = \bigcap_{\mathcal{I} \in \mathcal{M}} \mathcal{I}$ is a model of P.* □

Corollary 2.9 (Least Model) *Every homogeneous program P has a least Θ-model, N_P.* □

This Θ-model is the intended representant of all Θ-models of P in the following sense.

Corollary 2.10 *Let $\mathbf{A}$ be a conjunct and η be a substitution. Then $N_P \models \langle \mathbf{A}, \eta \rangle$ iff for all Θ-models $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$.* □

In the theory of Logic Programming the least Herbrand model can be generated as the least fixpoint of the immediate consequence operator T_P on the Herbrand interpretations. This characterization is useful for establishing the completeness of SLD-resolution. We now provide an analogous characterization of the least Θ-model N_P in order to show the completeness of the LD-resolution.

First, we introduce the appropriate operator T_P.

Definition 2.11 Let P be a homogeneous program. The immediate consequence operator T_P on the Θ-interpretations is defined as follows:

$$T_P(\mathcal{I}) = \{\langle H\theta, \eta \,|\, H\theta\rangle \mid \begin{array}{l}\text{for some } \mathbf{B} \\ H \leftarrow \mathbf{B} \text{ is a variant of a clause from } P, \\ Dom(\theta) = Var(H),\ Ran(\theta) \cap Var(\mathbf{B}) = \emptyset, \\ \mathcal{I} \models \langle \mathbf{B}\theta, \eta\rangle\}.\end{array}$$

□

Next, we characterize the Θ-models of P as the pre-fixpoints of T_P.

Lemma 2.12 (Model Characterization) *$\mathcal{I}$ is a Θ-model of P iff $T_P(\mathcal{I}) \subseteq \mathcal{I}$.*

Proof. The T_P operator is easily seen to be additive, i.e. for every Θ-interpretation $\mathcal{I}$ we have $T_{P\cup P'}(\mathcal{I}) = T_P(\mathcal{I}) \cup T_{P'}(\mathcal{I})$.

Thus it suffices to prove the claim when P consists of just one clause, c. Then for every H, θ and η we have $\langle H\theta, \eta \,|\, H\theta\rangle \in T_{\{c\}}(\mathcal{I})$ iff (by Definition 2.11)

$H \leftarrow \mathbf{B}$ is a variant of c such that $\mathcal{I} \models \langle \mathbf{B}\theta, \eta\rangle$, $Dom(\theta) = Var(H)$ and $Ran(\theta) \cap Var(\mathbf{B}) = \emptyset$. Since $\mathcal{I}$ is a model of $\{c\}$ then this holds iff $\mathcal{I} \models \langle H\theta, \eta \,|\, (H\theta)\rangle$, i.e. $\langle H\theta, \eta \,|\, (H\theta)\rangle \in \mathcal{I}$. □

Now, we characterize N_P as the least fixpoint of T_P. We need the following observation.

Proposition 2.13 (Monotonicity) *T_P is monotonic, that is $I \subseteq J$ implies $T_P(I) \subseteq T_P(J)$.*

□

Proposition 2.14 (Least Fixpoint) *T_P has a least fixpoint $lfp(T_P)$ which is also its least pre-fixpoint.* □

We can now derive the desired result.

Corollary 2.15 *$lfp(T_P) = N_P$.*

Proof. By the Least Fixpoint Proposition 2.14, Least Model Lemma 2.9 and Model Characterization Corollary 2.12. □

Finally, we provide a more precise characterization of the Θ-model N_P that will be used in the proof of the completeness of the LD-resolution. We need the following strengthening of the Monotonicity Proposition 2.13.

Proposition 2.16 (Continuity) *T_P is continuous, that is for every sequence $\mathcal{I}_i$ $(i \geq 0)$ of Θ-interpretations such that $\mathcal{I}_0 \subseteq \mathcal{I}_1 \subseteq \ldots$ we have*

$$T_P(\cup_{i=0}^{\infty}\mathcal{I}_i) = \cup_{i=0}^{\infty} T_P(\mathcal{I}_i).$$

□

We define now a sequence of Θ-interpretations by

$$T_P \uparrow 0 = \emptyset,$$
$$T_P \uparrow (n+1) = T_P(T_P \uparrow n),$$
$$T_P \uparrow \omega = \cup_{i=0}^{\infty} T_P \uparrow i.$$

Proposition 2.17 (Characterization) $N_P = T_P \uparrow \omega$. □

We can now prove the completeness of LD-resolution with respect to the Θ-semantics for homogeneous programs.

Theorem 2.18 (Completeness I) *Consider a homogeneous program P and a conjunct $\mathbf{A}$. Suppose that for all Θ-models $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$. Then there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{A} \}$ with c.a.s. η.*

Proof. In particular we have $N_P \models \langle \mathbf{A}, \eta \rangle$. By the Characterization Proposition 2.17 $T_P \uparrow \omega \models \langle \mathbf{A}, \eta \rangle$. By the monotonicity of T_P we have $T_P \uparrow 0 \subseteq T_P \uparrow 1 \subseteq \ldots$, so by the Continuity Lemma 2.16 $T_P \uparrow k \models \langle \mathbf{A}, \eta \rangle$ for some $k > 0$.

We now prove the claim by induction w.r.t. the lexicographic ordering $<$ defined on pairs $\langle k, l(\mathbf{A}) \rangle$ of natural numbers. In this ordering

$$\langle n_1, n_2 \rangle < \langle m_1, m_2 \rangle \text{ iff } n_1 < m_1 \text{ or } (n_1 = m_1 \wedge n_2 < m_2).$$

The case when $\mathbf{A}$ is empty, i.e. $l(\mathbf{A}) = 0$ (which covers the base case of the induction) is immediate by Definition 2.6.

Suppose now $\mathbf{A} = A, \mathbf{B}$. There exist substitutions θ, σ such that

$$T_P \uparrow k \models \langle A, \theta \rangle,$$

$$T_P \uparrow k \models \langle \mathbf{B}\theta, \sigma \rangle,$$

$(A, \mathbf{B}, \theta, \sigma)$ is a good tuple and $\eta = (\theta\sigma) | (A, \mathbf{B})$.

We first prove that $P \cup \{ \leftarrow A \}$ has an LD-refutation with c.a.s. θ. When A is a built-in atom this conclusion follows immediately from Definitions 2.1 and 2.6.

When A is defined in P we have $k > 0$. By Definition 2.11 there exists a variant $H \leftarrow \mathbf{B}'$ of a clause from P, a substitution ψ s.t. $Dom(\psi) = Var(H)$, $Ran(\psi) \cap Var(\mathbf{B}') = \emptyset$, $A = H\psi$ and a substitution ϕ such that $T_P \uparrow (k-1) \models \langle \mathbf{B}'\psi, \phi \rangle$ and $\theta = \phi | A$.

Since $\langle k-1, l(\mathbf{B}'\psi) \rangle < \langle k, l(\mathbf{A}) \rangle$, by the induction hypothesis there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{B}'\psi \}$ with c.a.s. ϕ. Therefore there exists an LD-refutation of $P \cup \{ \leftarrow A \}$ with c.a.s. θ, because $\leftarrow \mathbf{B}'\psi$ is a resolvent of $\leftarrow A$ using the mgu ψ and (since $A\psi = A$) $\theta = (\psi\phi) | A$.

Since $\langle k, l(\mathbf{B}\theta) \rangle < \langle k, l(\mathbf{A}) \rangle$, by the induction hypothesis also there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{B}\theta \}$ with c.a.s. σ. Since $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple and $\eta = (\theta\sigma) | A, \mathbf{B}$, we can apply the Good Tuple Lemma 2.4. We conclude that there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{A} \}$ with c.a.s. η. □

Corollary 2.19 *Let P be a homogeneous program. Then*

$$N_P = \{ \langle A, \eta \rangle \mid A \text{ is defined in } P \text{ and there exists an LD-refutation of } P \cup \{ \leftarrow A \} \text{ with c.a.s. } \eta \}.$$

Proof. By Definition 2.6, and Theorems 2.7 and 2.18. □

Now, every program can be easily transformed into a homogeneous program with the same computational behaviour.

Definition 2.20 (Homogeneous Form) Let P be a Prolog program. Let $x_1, x_2, \ldots$ be distinct variables not occurring in P. Transform each clause

$$p(t_1, \ldots, t_k) \leftarrow \mathbf{B}$$

of P into the clause

$$p(x_1, \ldots, x_k) \leftarrow x_1 = t_1, \ldots, x_k = t_k, \mathbf{B}.$$

Here "=" is the previously defined built-in and interpreted as "is unifiable with". We denote the resulting program by $Hom(P)$ and call it a *homogeneous form* of P. □

A Prolog program P and its homogeneous form $Hom(P)$ have the same computational behaviour.

Theorem 2.21 *Let P be a Prolog program and G a goal. Then $P \cup \{G\}$ has an LD-refutation with c.a.s. η iff $Hom(P) \cup \{G\}$ has an LD-refutation with c.a.s. η.* □

This allows us to reason about the meaning of Prolog programs by transforming them first to a homogeneous form. Alternatively, we can extend the definition of the truth to arbitrary programs by simply defining a clause to be true iff its homogeneous version is true. By "processing" then the meaning of the introduced calls to the built-in "=" we obtain a direct definition of truth of a clause. Due to space limitations we do not present here these results and refer the interested reader to the full version of the paper.

3 Termination of Prolog Programs

In this section we show that the Θ-semantics is helpful when studying termination of Prolog programs. The presence of built-in's allows us to better control the execution of the programs and consequently it is not surprising that most "natural" programs with built-in's terminate for *all* goals. This motivates the following definition.

Definition 3.1 We say that a Prolog program P is *strongly terminating* if for all goals G, all LD-derivations of $P \cup \{G\}$ are finite. □

Traditionally, the main concept used to prove termination of Prolog programs is that of a level mapping. In our case it is convenient to allow level mappings defined on non-ground atoms and yielding values in a well-founded ordering.

Definition 3.2 A *level mapping* $|\ |$ is a function from atoms to a well-founded ordering such that $|A| = |B|$ if A and B are variants. □

The following auxiliary notion will be used below.

Definition 3.3 c' is called a *head instance* of a clause c if $c' = c\theta$ for some substitution that instantiates only variables of c that appear in its head. □

First we provide a method for proving (strong) termination of homogeneous programs. Our key concept in establishing termination is the following one.

Definition 3.4 A homogeneous program P is called *acceptable* w.r.t. a *level mapping* $|\ |$ and a Θ-*model* I *of* P if for all head instances $A \leftarrow B_1, \ldots, B_n$ of a clause of P, the following implication holds for $i \in [1, n]$:

if $I \models \langle B_1, \ldots, B_{i-1}, \eta \rangle$ then $|A| > |B_i\eta|$.

P is called *acceptable* if it is acceptable w.r.t. some level mapping and a Θ-model of P. □

The relevance of the notion of acceptability is clarified by the following theorem.

Theorem 3.5 (Soundness II) *Let P be a homogeneous program. Suppose P is acceptable. Then P is strongly terminating.*

The following notion will be useful in the proof.

Definition 3.6

- By the *length* of a goal we mean the number of its atoms. For a goal G we denote its length by $l(G)$.
- Consider an LD-derivation ξ. Let G be a goal in ξ. Let k be the minimum length of a goal in the suffix of ξ starting at G and let H be the first goal in this suffix with length k. We call H *the shortest goal of* ξ *under* G. □

Proof of Theorem 3.5.

Suppose by contradiction that there exists an infinite LD-derivation of $P \cup \{G\}$. Call it ξ. Denote G by H_0. We first define two infinite sequences $G_1, G_2, \ldots$ and $H_1, H_2, \ldots$ of goals of ξ by the following formula for $j \geq 1$:

G_j is the shortest goal of ξ under H_{j-1},
H_j is the direct descendant of G_j in ξ.

Fix $j \geq 1$. Let $A \leftarrow B_1, \ldots, B_n$ be the input clause and θ the mgu used to obtain H_j from G_j. By the choice of G_j and H_j we have $l(G_j) \leq l(H_j)$, so $n \geq 1$. G_j is of the form $\leftarrow C_1, \ldots, C_k$ where $k \geq 1$ and H_j is of the form $\leftarrow (B_1, \ldots, B_n, C_2, \ldots, C_k)\theta$. By definition, no goal of ξ under G_j is of length less than k, so G_{j+1} is of the form $\leftarrow (B_i, \ldots, B_n, C_2, \ldots, C_k)\theta\eta$ for some η, where $i \in [1, n-1]$. This means that there exists an LD-refutation of $P \cup \{\leftarrow (B_1, \ldots, B_{i-1})\theta\}$ with c.a.s. η. This refutation is obtained by deleting from all goals of ξ between and including H_j and G_{j+1} all occurrences of the instantiated versions of $B_i\theta, \ldots, B_n\theta, C_2\theta, \ldots, C_n\theta$.

By the Soundness Theorem 2.7 we have $I \models \langle (B_1, \ldots, B_{i-1})\theta, \eta \rangle$. By the acceptability of P

$$|A\theta| > |B_i\theta\eta|. \tag{6}$$

By the assumption stated at the beginning of Section 2.3 the mgu μ used to obtain H_{j+1} from G_{j+1} does not bind the variables of the selected atom $B_i\theta\eta$. Thus $B_i\theta\eta = B_i\theta\eta\mu$ and consequently

$$|B_i\theta\eta| = |B_i\theta\eta\mu|. \tag{7}$$

So, assuming $j > 1$, we have

$$|C_1| = |C_1\theta|, \tag{8}$$

(C_1 is the first atom of G_j and $B_i\theta\eta$ is the first atom of G_{j+1}). But θ unifies A and C_1, so

$$|C_1\theta| = |A\theta|. \tag{9}$$

By (6), (8), and (9) we conclude, assuming $j > 1$,

$$|C_1| > |B_i\theta\eta|.$$

Thus applying the level mapping $|\ |$ to the first atoms of the goals $G_2, G_3, \ldots$ we obtain an infinite descending sequence of elements of a well-founded ordering. This yields a contradiction. □

We now prove a converse of the Soundness II Theorem 3.5. For a strongly terminating Prolog program P and a goal G, we denote by $nodes_P(G)$ the number of nodes in the LD-tree of $P \cup \{G\}$. The following lemma summarizes the relevant properties of $nodes_P(G)$.

Lemma 3.7 (LD-tree) *Let P be a strongly terminating Prolog program. Then*

(i) $nodes_P(G) = nodes_P(H)$ if G and H are variants,

(ii) $nodes_P(H) < nodes_P(G)$ for all non-root nodes H in the LD-tree of $P \cup \{G\}$,

(iii) $nodes_P(H) \leq nodes_P(G)$ for all prefixes H of G.

Proof. (i) By a simple generalization of the Variant Lemma 2.8 of Apt [Apt90] to the class of Prolog programs, an isomorphism between the LD-trees of $P \cup \{G\}$ and $P \cup \{H\}$ can be established.
(ii), (iii) Immediate by the definition. □

We are now in position to prove the desired result.

Theorem 3.8 (Completeness II) *Let P be a homogeneous program. Suppose P is strongly terminating. Then P is acceptable.*

Proof. Put for an atom A

$$|A| = nodes_P(A).$$

By Lemma 3.7 (i) $|\ |$ is a level mapping. We now prove that P is acceptable w.r.t. $|\ |$ and N_P, the least Θ-model of P. To this end consider a clause c with head A_0 and its head instance $c\theta = A \leftarrow B_1, \ldots, B_n$ where $Dom(\theta) \subseteq Var(A_0)$. Let us assume that $c\theta$ is disjoint with c. Then A is disjoint with A_0, $A = A_0\theta$ and $Dom(\theta) \subseteq Var(A_0)$, so θ is idempotent and $A\theta = A$. Thus θ unifies A and A_0 and it is easy to see that in fact θ is an mgu of A and A_0. Thus $\leftarrow B_1, \ldots, B_n$ is a resolvent of $\leftarrow A$ with the input clause c. By Lemma 3.7 (ii)

$$nodes_P(\leftarrow A) > nodes_P(\leftarrow B_1, \ldots, B_n). \tag{10}$$

This conclusion was reached under the assumption that $c\theta$ is disjoint with c but Lemma 3.7 (i) allows us to dispense us with this assumption. Suppose now that $N_P \models \langle B_1, \ldots, B_{i-1}, \eta\rangle$ for some $i \in [1, n]$ and substitution η. Then by the Completeness Theorem 2.18 there exists an

LD-refutation of $\leftarrow B_1, \ldots, B_{i-1}$ with c.a.s. η, so $\leftarrow (B_i, \ldots, B_n)\eta$ is a node in the LD-tree of $P \cup \{\leftarrow B_1, \ldots, B_n\}$. By Lemma 3.7 (ii)

$$nodes_P(\leftarrow B_1, \ldots, B_n) \geq nodes_P(\leftarrow (B_i, \ldots, B_n)\eta) \tag{11}$$

and by Lemma 3.7 (iii)

$$nodes_P(\leftarrow (B_i, \ldots, B_n)\eta) \geq nodes_P(\leftarrow B_i\eta). \tag{12}$$

By (10), (11), and (12) we now conclude

$$nodes_P(\leftarrow A) > nodes_P(\leftarrow B_i\eta),$$

i.e. $|A| > |B_i\eta|$.

This shows that P is acceptable. □

This establishes equivalence between the notions of acceptability and strong termination for homogeneous programs. For arbitrary programs we note the following result.

Theorem 3.9 *Let P be a Prolog program and G a goal. Then the LD-tree of $P \cup \{G\}$ is finite iff the LD-tree of $Hom(P) \cup \{G\}$ is finite.* □

Corollary 3.10 *Let P be a Prolog program. Then P strongly terminates iff $Hom(P)$ strongly terminates.* □

This allows us to reason about termination of Prolog programs by transforming them first to a homogeneous form and then using the notion of acceptability. An alternative, direct way of reasoning about termination can be found in the full version of the paper.

The introduction of homogeneous programs allows us to draw the following conclusion.

Theorem 3.11 *Let P be a Prolog program. Then P strongly terminates iff $Hom(P)$ is acceptable.*

Proof. By the Soundness I Theorem 3.5 and Completeness I Theorem 3.8 applied to $Hom(P)$ and Corollary 3.10. □

Finally we show how this approach to termination can be modularized. First, we need a notion of an extension.

Definition 3.12 We say that a relation p is *defined* in a Prolog program P if p occurs in a head of a clause from P. □

Definition 3.13 Let P_1 and P_2 be two Prolog programs. We say that P_2 *extends* P_1, and write $P_1 < P_2$, if

- P_1 and P_2 define different relations,
- no relation defined in P_2 occurs in P_1. □

Informally, P_2 extends P_1 if P_2 defines *new* relations, possibly using the relations defined already in P_1. The following theorem formalizes our modular approach to termination.

Theorem 3.14 (Modularity) *Suppose P_2 extends P_1. Assume that*

(i) P_1 is acceptable,

(ii) P_2 is acceptable w.r.t. a Θ-model I of $P_1 \cup P_2$ and a level mapping $|\ |$ such that $|A| = 0$ if A contains a relation defined in P_1.

Then $P_1 \cup P_2$ is strongly terminating.

Proof. P_2 extends P_1. Thus $P_1 \cup P_2$ is strongly terminating iff P_1 is strongly terminating and P_2 is strongly terminating when the relations defined in P_1 are treated as built-in's defined by

$$[\![p]\!] = \{\langle A, \eta\rangle \mid A \text{ contains } p \text{ and there exists an LD-refutation of } P_1 \cup \{\leftarrow A\} \text{ with c.a.s. } \eta\ \}.$$

Now, by (i) and the Soundness I Theorem 3.5 P_1 is strongly terminating. To deal with the other conjunct consider $N_{P_1 \cup P_2}$, the least Θ-model of $P_1 \cup P_2$. By (ii) and Corollary 2.10 P_2 is acceptable w.r.t. $N_{P_1 \cup P_2}$ and the level mapping $|\ |$. Moreover, by Corollary 2.19 and the fact that P_2 extends P_1 we have for all atoms A containing a relation p defined in P_1

$$N_{P_1 \cup P_2} \models \langle A, \eta\rangle \text{ iff } \langle A, \eta\rangle \in [\![p]\!].$$

Thus by the Soundness I Theorem 3.5 P_2 is strongly terminating when the relations defined in P_1 are treated as built-in's defined as above.

This concludes the proof of the theorem. □

We applied this approach to prove termination of the `list` program from the introduction, of the typed version of the append program and of both versions of the `unify` program of Sterling and Shapiro [SS86]. Modularity Theorem 3.14 allowed us to present these proofs in a modular way, by proving termination of various program parts separatly.

We believe that the approach to the semantics and termination presented here can be extended to general programs, i.e. programs admitting negative literals in the body. To this end some of the ideas of Apt and Pedreschi [AP91] could be of use.

Acknowledgements. We thank Annalisa Bossi and Kees Doets for helpful discussions on the subject of the Good Tuple Lemma 2.4 and the referees for useful suggestions concerning the presentation.

References

[AP91] K. R. Apt and D. Pedreschi. Proving termination of general Prolog programs. In T. Ito and A. Meyer, editors, *Proceeding of the International Conference on Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 526, pages 265–289, Berlin, 1991. Springer-Verlag.

[Apt90] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier, 1990. Vol. B.

[Bör89] E. Börger. A logical operational semantics of full Prolog, Part III: Built-in predicates for files, terms, arithmetic and input-output. In Y.Moschovakis, editor, *Proceedings Workshop on Logic from Computer Science*. Springer MSRI Publications, 1989.

[Cla79] K.L. Clark. Predicate logic as a computational formalism. Res. Report DOC 79/59, ico, London, 1979.

[DF87] P. Deransart and G. Ferrand. An operational formal definition of Prolog. In *Proceedings of the 4th Symposium on Logic Programming*, pages 162–172. Computer Society Press, 1987.

[HL88] P.M. Hill and J.W. Lloyd. Analysis of meta-programs. In H.D. Abramson and M.H. Rogers, editors, *Proceedings of the Meta88 Workshop*, pages 23–52. MIT Press, 1988.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[MP89] M. Martelli M.Falaschi, G. Levi and C. Palamidessi. Declarative modeling of the operational behaviour of logic languages. *Theoretical Computer Science*, 69:289–318, 1989.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, 1986.

[vEK76] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:733–742, 1976.